

**Howto - private PKI system just with
OpenSSL**

YAPKI

-

Yet Another PKI

Index

1 General information's about Certificates.....	4
1.1 What is in for me with Certificates.....	4
1.2 What is SSL and what are Certificates?	4
1.2.1 Private Key/Public Key.....	5
1.2.2 The Certificate.....	5
1.2.3 The Symmetric key.....	5
1.2.4 Encryption algorithm.....	5
1.2.5 The Hash.....	6
1.2.6 Signing.....	6
1.2.7 Pass Phrase.....	6
1.2.8 X.509.....	6
1.3 What about S/Mime or other protocols?	6
2 Certificate Management with OpenSSL.....	7
2.1 Installation	7
2.1.1 System overview.....	7
2.1.2 YAPKI setup	7
2.1.3 The openssl.cnf file	8
2.2 Create your own CA (Certification Authority).....	11
2.2.1 Create the Certification Authority.....	11
2.2.2 Install the CA root certificate as a Trusted Root Certificate	11
2.3 Certificate administration.....	12
2.3.1 The index.txt file (the database).....	12
2.3.2 Display a certificate	12
2.3.3 Generate and Sign a certificate request	12
2.3.4 Renew a certificate	13
2.3.5 Revoke Certificate and create a CRL.....	13
2.3.5.1 Revoke a certificate	13
2.3.5.2 Create a CRL.....	13
2.3.6 Convert from A to B.....	14
2.3.7 Backup / Restore.....	15
2.4 Certificate Administration using a simple web interface.....	15
3 Using Certificates in web servers, email, browser and any other software.....	16
3.1 Securing your web server.....	16
3.1.1 Generate Certificate for web server.....	16
3.1.2 Setup & Customization.....	16
3.1.3 Backup / Restore.....	17
3.2 Securing Browsers (as https clients).....	17
3.2.1.1 Firefox, Opera and Mozilla.....	17
3.2.1.2 Internet Explorer.....	17
3.3 Securing E-mails.....	18
3.3.1 Generate and use an s/mime certificate	18
3.3.2 Using Google Mail.....	18
3.3.3 Mozilla Thunderbird.....	18
3.3.4 MS Outlook	18
3.4 Securing your data with PGP.....	19
3.5 Securing shell and file transfer.....	19
3.5.1 Openssh.....	19
3.5.2 Putty.....	20
3.6 Securing a VPN.....	20
3.7 Securing Code	21

3.7.1 Micosoft Code	21
3.7.2 Java Code.....	21
3.8 Securing Internet Protocols.....	21
3.8.1 Using a certificate with mod_ssl in apache.....	21
3.8.2 Using a certificate with IMAPS / POPS / SMTP.....	21
3.8.3 Using a certificate with Stunnel.....	22
3.8.4 Generate and Sign a key with Microsoft Key Manager	22
4 Glossary.....	23
5 The YAPKI system.....	24
5.1 Default directory layout.....	24
5.2 YAPKI scripts.....	24
6 Document release notes.....	25

Change Management

Change comment	Version ID
Initial version	1.0, 2002
Creation and release as part of yafra.org under Apache License Version 2	PKI-OpenSSL.odt 1 2010
Updated content and structure, added glossary	PKI-OpenSSL.odt 21 2011-01-30 18:37:35Z
Removed all fixe and updated information for mobile devices and other services	yafra.org git PKI-OpenSSL.odt 2013-09-15

License

yafra.org - YAPKI
Copyright [2005] Martin Weber, Switzerland

This product includes software developed at
The OpenSSL Project and from yafra.org.

Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1 General information's about Certificates

1.1 What is in for me with Certificates

Identification / Authentication:

The persons / entities with whom we are communicating are really who they say they are.

Confidentiality:

The information within the message or transaction is kept confidential. It may only be read and understood by the intended sender and receiver.

Integrity:

The information within the message or transaction is not tampered accidentally or deliberately with en route without all parties involved being aware of the tampering.

Non-Repudiation:

The sender cannot deny sending the message or transaction, and the receiver cannot deny receiving it.

Access Control:

The intended person or entity only realizes access to the protected information.

All the above security properties can be achieved and implemented through the use of Public Key Infrastructure (in particular Digital Certificates).

1.2 What is SSL and what are Certificates?

The Secure Socket Layer protocol was created by Netscape to ensure secure transactions between web servers and browsers. The protocol uses a third party, a Certificate Authority (CA), to identify one end or both end of the transactions. This is in short how it works.

- A browser requests a secure page (usually https://).
- The web server sends its public key with its certificate.
- The browser checks that the certificate was issued by a trusted party (usually a trusted root CA), that the certificate is still valid and that the certificate is related to the site contacted.
- The browser then uses the public key, to encrypt a random symmetric encryption key and sends it to the server with the encrypted URL required as well as other encrypted http data.
- The web server decrypts the symmetric encryption key using its private key and uses the symmetric key to decrypt the URL and http data.
- The web server sends back the requested html document and http data encrypted with the symmetric key.
- The browser decrypts the http data and html document using the symmetric key and displays the information.

Several concepts have to be understood here.

1.2.1 Private Key/Public Key

The encryption using a private key/public key pair ensures that the data can be encrypted by one key but can only be decrypted by the other key pair. This is sometime hard to understand, but believe me it works. The keys are similar in nature and can be used alternatively: what one key encrypts, the other key pair can decrypt. The key pair is based on prime numbers and their length in terms of bits ensures the difficulty of being able to decrypt the message without the key pairs. The trick in a key pair is to keep one key secret (the private key) and to distribute the other key (the public key) to everybody. Anybody can send you an encrypted message, that only you will be able to decrypt. You are the only one to have the other key pair, right? In the opposite, you can certify that a message is only coming from you, because you have encrypted it with your private key, and only the associated public key will decrypt it correctly. Beware, in this case the message is not secured you have only signed it. Everybody has the public key, remember!

One of the problem left is to know the public key of your correspondent. Usually you will ask him to send you a non confidential signed message that will contains his public key as well as a certificate.

1.2.2 The Certificate

How do you know that you are dealing with the right person or rather the right web site. Well, someone has taken great length (if they are serious) to ensure that the web site owners are who they claim to be. This someone, you have to implicitly trust: you have his/her certificate loaded in your browser (a root Certificate). A certificate, contains information about the owner of the certificate, like e-mail address, owner's name, certificate usage, duration of validity, resource location or Distinguished Name (DN) which includes the Common Name (CN) (web site address or e-mail address depending of the usage) and the certificate ID of the person who certifies (signs) this information. It contains also the public key and finally a hash to ensure that the certificate has not been tampered with. As you made the choice to trust the person who signs this certificate, therefore you also trust this certificate. This is a certificate trust tree or certificate path. Usually your browser or application has already loaded the root certificate of well known Certification Authorities (CA) or root CA Certificates. The CA maintains a list of all signed certificates as well as a list of revoked certificates. A certificate is insecure until it is signed, as only a signed certificate cannot be modified. You can sign a certificate using itself, it is called a self signed certificate. All root CA certificates are self signed.

1.2.3 The Symmetric key

Well, Private Key/Public Key encryption algorithms are great, but they are not usually practical. It is asymmetric because you need the other key pair to decrypt. You can't use the same key to encrypt and decrypt. An algorithm using the same key to decrypt and encrypt is deemed to have a symmetric key. A symmetric algorithm is much faster in doing its job than an asymmetric algorithm. But a symmetric key is potentially highly insecure. If the enemy gets hold of the key then you have no more secret information. You must therefore transmit the key to the other party without the enemy getting its hands on it. As you know, nothing is secure on the Internet. The solution is to encapsulate the symmetric key inside a message encrypted with an asymmetric algorithm. You have never transmitted your private key to anybody, then the message encrypted with the public key is secure (relatively secure, nothing is certain except death and taxes). The symmetric key is also chosen randomly, so that if the symmetric secret key is discovered then the next transaction will be totally different.

1.2.4 Encryption algorithm

There are several encryption algorithms available, using symmetric or asymmetric methods, with keys of various lengths. Usually, algorithms cannot be patented, if Henri Poincare had patented his algorithms, then he would have been able to sue Albert Einstein... So algorithms cannot be patented except mainly in USA. OpenSSL is developed in a country where algorithms cannot be patented and where encryption technology is not reserved to state agencies like military and secret services. During the negotiation between browser and web server, the applications will indicate to each other a list of algorithms that can be understood ranked by order of preference.

The common preferred algorithm is then chosen. OpenSSL can be compiled with or without certain algorithms, so that it can be used in many countries where restrictions apply.

1.2.5 The Hash

A hash is a number given by a hash function from a message. This is a one way function, it means that it is impossible to get the original message knowing the hash. However the hash will drastically change even for the slightest modification in the message. It is therefore extremely difficult to modify a message while keeping its original hash. It is also called a message digest. Hash functions are used in password mechanisms, in certifying that applications are original (MD5 sum), and in general in ensuring that any message has not been tampered with. It seems that the Internet Engineering Task Force (IETF) prefers SHA1 over MD5 for a number of technical reasons (Cf RFC2459 7.1.2 and 7.1.3).

1.2.6 Signing

Signing a message, means authenticating that you have yourself assured the authenticity of the message (most of the time it means you are the author, but not necessarily). The message can be a text message, or someone else's certificate. To sign a message, you create its hash, and then encrypt the hash with your private key, you then add the encrypted hash and your signed certificate with the message. The recipient will recreate the message hash, decrypts the encrypted hash using your well known public key stored in your signed certificate, check that both hash are equals and finally check the certificate.

The other advantage of signing your messages is that you transmit your public key and certificate automatically to all your recipients.

1.2.7 Pass Phrase

A pass phrase is like a password except it is longer. In the early days passwords on Unix system were limited to 8 characters, so the term pass phrase for longer passwords. Longer is the password harder it is to guess. Nowadays Unix systems use MD5 hashes, which have no limitation in length of the password.

1.2.8 X.509

The X.509 (see ITU recommendation <http://www.itu.int/rec/T-REC-X.509> or see Wikipedia <http://en.wikipedia.org/wiki/X.509>) standard defines amongst other things the format for digital certificates. A digital certificate may be stored in different formats depending on its purpose. You will hear names like PEM, DER, PKCS and others. See as well the chapter Glossary. See as well <http://www.ietf.org/rfc/rfc5280.txt>.

1.3 What about S/Mime or other protocols?

If SSL was developed for web servers, it can be used to encrypt any protocol. Any protocol can be encapsulated inside SSL. This is used in IMAPS, POPS, SMTPS,... These secure protocols will use a different port than their insecure version. SSL can also be used to encrypt any transaction: there is no need to be in direct (live) contact with the recipient. S/Mime is such protocol; it encapsulates an encrypted message inside a standard e-mail. The message is encrypted using the public key of the recipient. If you are not online with the recipient then you must know its public key. Either you get it from its web site, from a repository, or you request the recipient to e-mail you its public key and certificate (to ensure you are speaking to the right recipient).

In a reverse order, the browser can send its own-signed certificate to the web server, as a mean of authentication. But everybody can get the browser certificate on the CA web site. Yes, but the signed certificate has been sent encrypted with the private key that only the public key can decrypt.

2 Certificate Management with OpenSSL

2.1 Installation

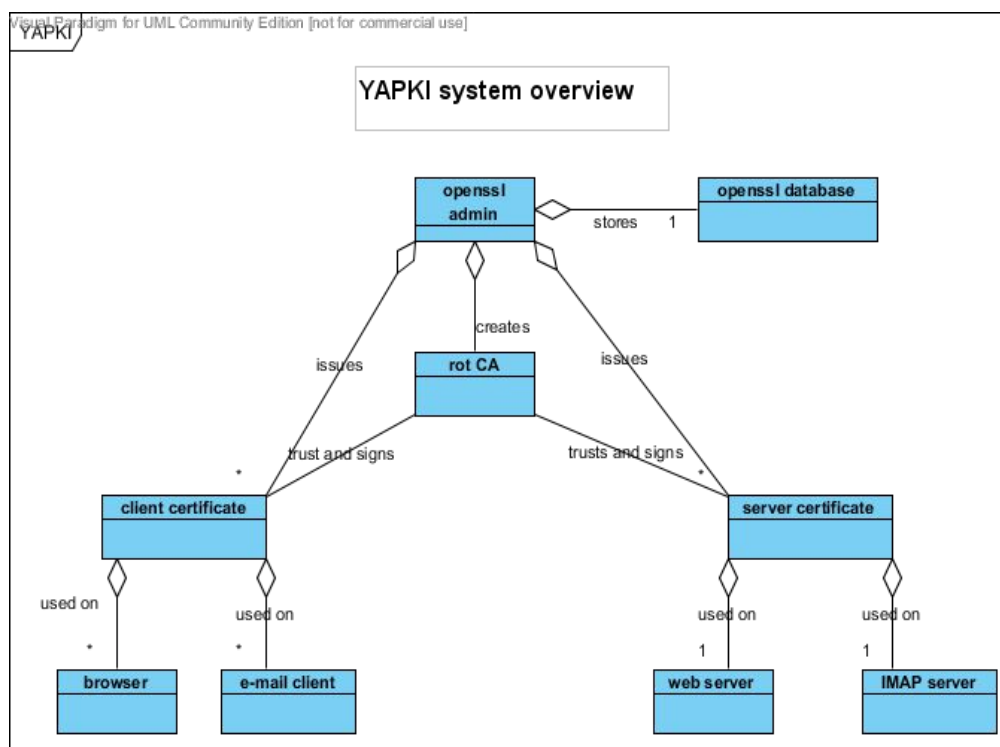
Nowadays, you do not have to worry too much about installing OpenSSL: most distributions use package management applications. Refer to your distribution documentation, or read the README and INSTALL file inside the OpenSSL tar ball.

Basically I tested the openssl distributions for win32, linux and cygwin. All work very fine. You can even reuse/transfer the configuration files easily. Every detailed description is based on the operating setup described in the attachment.

Some important points to consider:

- Set the time to a synchronization time server (NTP)

2.1.1 System overview



2.1.2 YAPKI setup

YAPKI is based on a Ubuntu Server operating system. Microsoft Windows is of course as well an option, but I don't use it for this purpose. Scripts are all working for bash, however most are available as batch files as well.

Create somewhere on your disc (where you don't have a file share activated) a ssl directory (which I will refer as **SSL_ROOT**). Below this directory we will setup the ca directory (which I will refer as **CA_ROOT**) and install some scripts.

Install scripts to the SSL_ROOT.

YAPKI - How to make your own PKI system with OpenSSL

Ensure that the utility CA.pl is in an accessible directory such as /usr/sbin. CA.pl can be found inside /usr/lib/ssl directories. CA.pl is a utility that hides the complexity of the openssl command.

```
$$$SLEAY_CONFIG="-config SSL_ROOT/openssl.cnf";
$DAYS="-days 365";      # 1 year
$CADAYS="-days 3650";   # 10 years

$CATOP="CA_ROOT";
```

Add this text and delete `--create_serial` to start off with 01:

```
open OUT, ">${CATOP}/serial";
print OUT "01";
close OUT;
```

2.1.3 The openssl.cnf file

Your openssl.cnf must be configured accordingly to minimize input entry.

```
#---Begin---
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
# RANDFILE = $ENV::HOME/.rnd oid_file = $ENV::HOME/.oid oid_section = new_oids
# To use this configuration file with the "-extfile" option of the
# "openssl x509" utility, name here the section containing the
# X.509v3 extensions to use:
# extensions =
# (Alternatively, use a configuration file that has only
# X.509v3 extensions in its main [= default] section.)
[ new_oids ]
# We can add new OIDs in here for use by 'ca' and 'req'.
# Add a simple OID like this:
# testoid1=1.2.3.4
# Or use config file substitution like this:
# testoid2=${testoid1}.5.6
##### [ ca ]
default_ca = CA_default # The default ca section
#####
[ CA_default ]
dir                = CA_ROOT                # Where everything is kept
certs              = $dir/certs             # Where the issued certs are kept
crl_dir            = $dir/crl               # Where the issued crl are kept
database           = $dir/index.txt        # database index file.
new_certs_dir      = $dir/newcerts         # default place for new certs.

certificate        = $dir/cacert.pem       # The CA certificate
serial            = $dir/serial            # The current serial number
crl               = $dir/crl.pem           # The current CRL
private_key        = $dir/private/akey.pem # The private key
RANDFILE          = $dir/private/.rand     # private random number file
x509_extensions    = usr_cert             # The extensions to add to the cert
# Extensions to add to a CRL. Note: Netscape communicator chokes on V2 CRLs #
# so this is commented out by default to leave a V1 CRL. # crl_extensions =
crl_ext
default_days       = 365                   # how long to certify for
default_crl_days   = 30                   # how long before next CRL
default_md         = sha1                 # which md to use.
preserve          = no                    # keep passed DN ordering
# A few difference way of specifying how similar the request should look # For
type CA, the listed attributes must be the same, and the optional # and
supplied fields are just that :- ) policy = policy_match
```


YAPKI - How to make your own PKI system with OpenSSL

```
# For the CA policy [ policy_match ] countryName = match stateOrProvinceName =
optional
localityName          = match
organizationName       = match
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional
# For the 'anything' policy # At this point in time, you must list all
acceptable 'object' # types. [ policy_anything ] countryName = optional
stateOrProvinceName = optional
localityName         = optional
organizationName      = optional
organizationalUnitName = optional
commonName           = supplied
emailAddress          = optional
##### [ req ]
default_bits = 1024 default_keyfile = privkey.pem distinguished_name =
req_distinguished_name
attributes    = req_attributes
default_md    = sha1
x509_extensions = v3_ca # The extensions to add to the self signed cert
[ req_distinguished_name ] countryName = Country Name (2 letter code)
countryName_default = FJ
countryName_min     = 2
countryName_max     = 2

stateOrProvinceName      = State or Province Name (full name)
stateOrProvinceName_default = Fiji
localityName = Locality Name (eg, city) localityName_default = Suva
0.organizationName = Organization Name (eg, company) 0.organizationName_default
= SOPAC
# we can do this but it is not needed normally :- ) #1.organizationName = Second
Organization Name (eg, company) #1.organizationName_default = World Wide Web
Pty Ltd
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = ITU
commonName = Common Name (eg, YOUR name) commonName_max = 64 emailAddress =
Email Address emailAddress_max = 40
# SET-ex3 = SET extension number 3
[ req_attributes ] challengePassword = A challenge password
challengePassword_min = 4 challengePassword_max = 20
unstructuredName = An optional company name
[ usr_cert ]
# These extensions are added when 'ca' signs a request. # This goes against
PKIX guidelines but some CAs do it and some software # requires this to avoid
interpreting an end user certificate as a CA.
basicConstraints=CA:FALSE
# Here are some examples of the usage of nsCertType. If it is omitted # the
certificate can be used for anything *except* object signing.
# This is OK for an SSL server. # nsCertType = server
# For an object signing certificate this would be used. # nsCertType = objsign
# For normal client use this is typical # nsCertType = client, email
# and for everything including object signing: # nsCertType = client, email,
objsign
# This is typical in keyUsage for a client certificate. # keyUsage =
nonRepudiation, digitalSignature, keyEncipherment
# This will be displayed in Netscape's comment listbox. nsComment =
"Certificate issued by https://www.sopac.org/ssl/"
# PKIX recommendations harmless if included in all certificates.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer:always
```

YAPKI - How to make your own PKI system with OpenSSL

```
# This stuff is for subjectAltName and issuerAltname. # Import the email
address. # subjectAltName=email:copy
# Copy subject details # issuerAltName=issuer:copy
# This is the base URL for all others URL addresses # if not supplied nsBaseUrl
= https://www.sopac.org/ssl/
# This is the link where to download the latest Certificate # Revocation List
(CRL) nsCaRevocationUrl = https://www.sopac.org/ssl/sopac-ca.crl
# This is the link where to revoke the certificate nsRevocationUrl =
https://www.sopac.org/ssl/revocation.html?
# This is the location where the certificate can be renewed nsRenewalUrl =
https://www.sopac.org/ssl/renewal.html?
# This is the link where the CA policy can be found nsCaPolicyUrl =
https://www.sopac.org/ssl/policy.html
# This is the link where we can get the issuer certificate issuerAltName =
URI:https://www.sopac.org/ssl/sopac.crt
# This is the link where to get the latest CRL crlDistributionPoints =
URI:https://www.sopac.org/ssl/sopac-ca.crl
[ v3_ca ]
# Extensions for a typical CA
# PKIX recommendation.
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid:always,issuer:always
# This is what PKIX recommends but some broken software chokes on critical #
extensions. # basicConstraints = critical,CA:true # So we do this instead.
basicConstraints = CA:true
# Key usage: this is typical for a CA certificate. However since it will #
prevent it being used as an test self-signed certificate it is best # left out
by default. # keyUsage = cRLSign, keyCertSign
# Some might want this also # nsCertType = sslCA, emailCA
# Include email address in subject alt name: another PKIX recommendation #
subjectAltName=email:copy # Copy issuer details # issuerAltName=issuer:copy
# RAW DER hex encoding of an extension: beware experts only! #
1.2.3.5=RAW:02:03 # You can even override a supported extension: #
basicConstraints= critical, RAW:30:03:01:01:FF
# This will be displayed in Netscape's comment listbox. nsComment =
"Certificate issued by https://www.sopac.org/ssl/"
# This is the base URL for all others URL addresses # if not supplied nsBaseUrl
= https://www.sopac.org/ssl/
# This is the link where to download the latest Certificate # Revocation List
(CRL) nsCaRevocationUrl = https://www.sopac.org/ssl/sopac-ca.crl
# This is the link where to revoke the certificate nsRevocationUrl =
https://www.sopac.org/ssl/revocation.html?
# This is the location where the certificate can be renewed nsRenewalUrl =
https://www.sopac.org/ssl/renewal.html?
# This is the link where the CA policy can be found nsCaPolicyUrl =
https://www.sopac.org/ssl/policy.html
# This is the link where we can get the issuer certificate issuerAltName =
URI:https://www.sopac.org/ssl/sopac.crt
# This is the link where to get the latest CRL crlDistributionPoints =
URI:https://www.sopac.org/ssl/sopac-ca.crl
[ crl_ext ] # CRL extensions. # Only issuerAltName and authorityKeyIdentifier
make any sense in a CRL. # issuerAltName=issuer:copy
authorityKeyIdentifier=keyid:always,issuer:always
#-----End-----
```

A few comments on openssl.cnf.

Variable names can use the suffixes `_default` for default value, `_min` for the minimum number of characters required and `_max` for the maximum number of characters required.

The file is composed of [Sections] of variables.

`dir:`

Specifies the base directory which I will refer to as CA_ROOT.

`default_ca:`

Specifies which section contains the variables for a default certificate.

`basicConstraints:`

Defines the usage of the certificate, for instance with CA:TRUE, the certificate is a root CA Certificate.

Add as well the OPENSSL_CONF environment variable and point it to your openssl.conf file. You do that on unix with:

```
export OPENSSL_CONF=SSL_ROOT/openssl.cnf
```

2.2 Create your own CA (Certification Authority)

2.2.1 Create the Certification Authority

Modify the openssl.cnf within section CA_default uncomment the `x509_extensions = v3_ca` and comment out (use "#") the `x509_extensions = usr_cert`. Make sure you change the CA.pl according to section "Installation".

To create a certification authority, use the command:

```
CA.pl -newca
```

On the first question you can press return to take the standard file names (that's fine for me). First you will be asked to set the CA key password (twice to verify). After that you have to fill out the questions. Depending on your configuration, you can just take the default. On questions like organization unit or e-mail or additional information I would recommend to use something like "CA-Admin".

After all questions you have to enter for the first time the CA key password, which is used to make the signature.

Convert now the PEM cacert.pem file into a DER file:

```
openssl x509 -in CA_ROOT/cacert.pem -out CA_ROOT/cacert.der -outform DER
```

and publish the certificates on your web server.

Now make sure that you created a "crlnumber" file within your CA_ROOT directory. You can put "01" as content. This is used to increment the id of the certification revocation list.

Do a backup now. See chapter Backup / Restore

There are a few requirements when you are a Certificate Authority (CA):

- You must publish your root CA Certificate, so that it can be widely installed in applications.
- You must publish the revocation list.
- You must display a certificate detail, provided its serial number
- You must provide a form for users to submit certificate requests.

All these requirements can be done using a web server and some scripting. Setting up a web server secured by SSL see in chapter Securing your web server.

2.2.2 Install the CA root certificate as a Trusted Root Certificate

Place this file on your web site as <http://mysite.com/ssl/cacert.crt>. Your web server should have a mime entry for .crt files. Your certificate is ready to be downloaded by any browser and saved.

It is important to publish the root CA Certificate on a web site as it is unlikely that people will have it already loaded on their browser. Beware, somebody could fake your web site and fake your root CA Certificate. If you can have more than one way for users to get your certificate, it is unlikely that a hacker will be able to corrupt everything.

2.3 Certificate administration

2.3.1 The index.txt file (the database)

In the index.txt file you can find the various certificate managed by OpenSSL.

Example certificate entry within index.txt:

```
V      981210145000Z      "leer"      01      unknown
/C=DE/O=Uni/OU=Inf/CN=TestCA/Email=testca@uni.de
```

The index file contains 6 columns sepearated by a tab (not through space/white space):

1. Status of certificate
2. Expiration date – syntax is YYMMDDHHMMSSZ
3. Empty for a valid certificate, else the date of revocation
4. Serial number of certificate (hex number)
5. Location of certificate – not used so it is "unknown"
6. Name of certificate holder, usually the Distinguished Name and E-Mail address.

The status of a certificate is marked with:

Certificate status	Description
R	Revoked
V	Valid
E	Expired

2.3.2 Display a certificate

You may have a certificate in its coded form, to read the details of the certificate just issue the following command:

```
openssl x509 -in newcert.pem -noout -text
```

The certificate structure is defined within X.509. See <http://www.itu.int/rec/T-REC-X.509/en>.

2.3.3 Generate and Sign a certificate request

```
CA.pl -newreq
```

or

```
openssl req -config openssl.cnf -new -keyout newreq.pem -out newreq.pem -days
365
```

creates a new private key and a certificate request and place it as newreq.pem. Enter a Common Name (CN) the main usage of the certificate for instance www.sopac.org if you want to secure the website www.sopac.org, or enter franck@sopac.org if you want to use to secure the e-mails of franck@sopac.org.

```
CA.pl -sign
```

or

```
openssl ca -config openssl.cnf -policy policy_anything -out newcert.pem
-infiles newreq.pem
```

will sign the request using the cacert.pem and commit the certificate as newcert.pem. You will need to enter the passphrase of the cacert.pem (your CA Certificate). The file newcerts/xx.pem will be created and index.txt and serial will be updated.

Your private key is in newreq.pem -PRIVATE KEY- and your certificate is in newcert.pem -CERTIFICATE-. A copy of newcert.pem is placed in newcerts/ with an adequate entry in index.txt so that a client can request this information via a web server to ensure the authenticity of the certificate.

Be aware of your newreq.pem file, because it contains a certificate request, but also your private key. The -PRIVATE KEY- section is not required when you sign it. So if you request someone else to sign your certificate request, ensure that you have removed the -PRIVATE KEY- section from the file. If you sign someone else certificate request, request from this person its -CERTIFICATE REQUEST- section not its private key.

2.3.4 Renew a certificate

The user sends you its old certificate request or create a new one based on its private key. First you have to revoke the previous certificate and sign again the certificate request. To find the old certificate, look in the index.txt file for the Distinguished Name (DN) corresponding to the request. Get the serial Number <xx>, and use the file cert/<xx>.pem as certificate for the revocation procedure.

You may want to sign the request manually because you have to ensure that the start date and end date of validity of the new certificate are correct.

```
openssl ca -config openssl.cnf -policy policy_anything -out newcert.pem -  
infile newreq.pem -startdate [now] -enddate [previous enddate+365days]
```

Replace [now] and [previous enddate+365days] by the correct values.

2.3.5 Revoke Certificate and create a CRL

2.3.5.1 Revoke a certificate

To revoke certificates simply issue the command:

```
openssl ca -revoke YOURCA/newcerts/03.pem
```

The 03.pem refers with "03" to the serial number from the index.txt database! The database is updated and the certificate is marked as revoked.

2.3.5.2 Create a CRL

You now need to generate the new revoked list of certificates. This Certificate Revocation List (CRL) file should be made available on your web site.

You may want to add the parameters crldays or crlhours and crlxts when you generate a CRL. The first two parameters indicate when the next CRL will be updated and the last one will use the crl_exts section in openssl.cnf to produce a CRL v2 instead of a CRL v1. The feature crl_exts is NOT used in this setup.

```
openssl ca -gencrl -config openssl.cnf -crldays 150 -out  
CA_ROOT/crl/yourcrlfile_no.crl
```

Or you could use the `gencrl.sh` / `gencrl.bat` file to do it.

Load this CRL into your browser and publish it to your webpage.

Beside the CRL you could also list the Certs, which are available and in production. So it is transparent who is a certified user/software. But that depends on your security policy.

2.3.6 Convert from A to B

The user must also be aware that OpenSSL supports several certificate formats. Certificates are based on the DSA signature algorithm and the RSA algorithm for public-key cryptography according to PKCS algorithms, as described in [7]. The certificate format depends on the application, as there is no agreement on file format standards. Private keys are usually available in the PEM and DER format. The related files have names of the following type:

- *key-rsa.pem for pem files
- *key-rsa.der for der files

For OpenSSL applications, the PEM format should suffice. For Java applications, the DER format might be more suitable for importing the private key and certificates. For certificates, the available formats are PEM, DER and PKCS12 with file names of the following type:

- *cert.pem for pem files
- *cert.der for der files
- *cert.p12 for pkcs12 files

In general, the PEM formats are mostly used in the Unix world, PCKS12 in the Microsoft world and DER in the Java world.

Certificate files are ASN.1-encoded objects that may be encrypted according to DES (Data Encryption Standard). The files can optionally be encrypted using a symmetric cipher algorithm, such as 3DES.

An unencrypted PEM file might look something like this:

```
-----BEGIN CERTIFICATE-----
MB4CGQDUoLoCULb9LsYm5+/WN992xxbiLQlEuIsCAQM=
-----END CERTIFICATE-----
```

The string beginning with MB4C... is the Base64-encoded, ASN.1-encoded object.

An encrypted file would have headers describing the type of encryption used, and the initialization vector:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-EDE3-CBC, C814158661DC1449
AFAZFbnQNrGjZJ/ZemdVSoZa3HWujxZuvBHZNoesxeyqqidFvnydA==
-----END RSA PRIVATE KEY-----
```

The two headers Proc-Type and DEK-Info declare the type of encryption, and the string starting with AFAZ... is the Base64-encoded, encrypted, ASN.1-encoded object.

Some PKCS#12 implementations require that a friendly name be specified using the name flag (for example, `-name "Friendly Name"`) which may be displayed on import.

There is an obsolete format called PFX which is incompatible and not to be confused with PKCS#12, even though Microsoft uses the '.PFX' extension in addition to '.P12' for PKCS#12 files.

Keys are sensitive information and should be stored carefully and encrypted using a strong pass phrase and cipher. You can use the DES, Triple DES, IDEA, or 128, 192, or 256 bit AES symmetric ciphers by adding a des, des3, idea, aes128, aes192, or aes256 flag to the command line. By default, keys will be encrypted with Triple DES.

If you do not have access to the pass phrase for an encrypted key it is unlikely you will be able to retrieve the key itself and will need to generate a new key and corresponding certificate(s).

As web browsers make use of Java applications, they import/export certificates in pkcs12 file format, i.e. public and private keys are packed in one single file using the PKCS#12 algorithm. Other applications require the pem format with unpacked public and private keys, thus the user must remember the appropriate file format for each application and must perform format conversions as appropriate

To convert from / to please look up this table:

Input format	Output format	Command
PEM	DER	<code>openssl x509 -in cert.pem -out cert.der -outform DER</code>
DER	PEM	<code>openssl x509 -in cert.der -inform DER -out cert.pem -outform PEM</code>
DER Key	PEM Key	<code>rsa -in input.key -inform DER -out output.key -outform PEM</code>
PEM	PKCS#12	<code>openssl pkcs12 -export -out cert.p12 -inkey userkey.pem -in usercert.pem</code> (with <code>-certfile root.crt</code> you could add a certificate up in the chain)
PKCS#12	PEM	<code>openssl pkcs12 -clcerts -nokeys -in cert.p12 -out usercert.pem</code> <code>openssl pkcs12 -nocerts -in cert.p12 -out userkey.pem</code>

2.3.7 Backup / Restore

Simply do a backup of your SSL_ROOT directory (a tar ball or with a backup tool). You could get some warning reading the random file, that's not an issue, just ignore, it will be generated on the fly anyway.

2.4 Certificate Administration using a simple web interface

Setup a plain Ubuntu Server with Apache installed. Install additional Apache module `mod_perl`. Install `openssl` and make sure this server hosts or has access to your `openssl` data files / database and to the YAPKI scripts.

Secure your web server, see chapter Securing your web server.

Within the SSL configuration of your Apache, set a `cgi-bin` directory.

Install the `cert.html` file into the SSL document root and the `pki-req.pl` into the `cgi-bin` directoy.

3 Using Certificates in web servers, email, browser and any other software

In all cases this consists of 2 steps for all use cases:

- You must install and trust your new CA
- You install the specific client or server based certificate and make sure the trust relationship to your CA is enabled and valid

3.1 Securing your web server

It is assumed that you have a recent Apache server installation up and running. The examples here refer to a recent plain Ubuntu Server installation with Apache installed.

3.1.1 Generate Certificate for web server

Just execute `genwwwcert.sh / genwwwcert.bat` and follow the instructions.

Important note: use as CN your FQDN, as this gets checked by your browser to see if the certificate CN matches with the FQDN you are connecting to. If this does not match, you will get warnings within your browser, normally the default action would be not connecting to this server.

Choose any password for the web server certificate (not one you use in other ways!!). The key will be modified in such a manner, that you don't have to enter the key on start up of the web server. If you want to enter the pass phrase upon start up, delete the last openssl command within the script.

Important note: If you remove the key so the web server does not ask you on start up, the key is not protected by a password – check the read permissions!

3.1.2 Setup & Customization

Under Ubuntu Server the configuration is available under `/etc/apache2`.

You need to make sure `mod_ssl` is enabled. Update the standard configuration files to enable the SSL site and to register your web server certificate. Within the SSL virtual server configuration you can grant access to anyone or to restrict access. Restricted access means that you list client certificate attributes like CN in your configuration and force to identify by a valid client certificate.

First generate and sign a certificate request with the Common Name (CN) as `www.mysite.com`. Remove any extra information to keep only the `---CERTIFICATE---` part.

The key needs to be made insecure, so no password is required when reading the private key. Take the `newreq.pem` files that contains your private key and remove the passphrase from it.

```
openssl rsa -in newreq.pem -out wwwkeyunsecure.pem
```

Because the key (PRIVATE Key) is insecure, you must know what you are doing: check file permissions, etc... If someone gets its hand on it, your site is compromised (you have been warned). Now you can use the `newcert` and `cakeyunsecure.pem` for apache. Copy `wwwkeyunsecure.pem` and `newcert.pem` in the directory `/etc/httpd/conf/ssl/` as `wwwkeyunsecure.pem` and `wwwcert.crt` respectively.

Edit `/etc/httpd/conf/ssl/ssl.default-vhost.conf`.

YAPKI - How to make your own PKI system with OpenSSL

```
# Server Certificate:
# Point SSLCertificateFile at a PEM encoded certificate. If
# the certificate is encrypted, then you will be prompted for a
# pass phrase. Note that a kill -HUP will prompt again. A test
# certificate can be generated with 'make certificate' under
# built time.
#SSLCertificateFile conf/ssl/ca.crt SSLCertificateFile wwwcert.crt
# Server Private Key:
# If the key is not combined with the certificate, use this
# directive to point at the key file.
#SSLCertificateKeyFile conf/ssl/ca.key.unsecure SSLCertificateKeyFile
wwwkeyunsecure.pem
```

Stop and start httpd (/etc/rc.d/init.d/httpd stop) ensure that all processes are dead (killall httpd) and start httpd (/etc/rc.d/init.d/httpd start)

3.1.3 Backup / Restore

Backup your httpd.conf, ssl.conf and all those ssl.xxx directories. Backup your http and https document and related directories (like cgi-bin).

3.2 Securing Browsers (as https clients)

3.2.1.1 Firefox, Opera and Mozilla

First of all you need to enable and trust your CA. Just drag and drop the cacert.der file into the browser and click on all the trust check boxes and commit it. Within the browser options you can specify for which type of application you trust this certificate: web site security, e-mail signing, code signing.

Go to your web page and follow the certification request page for Firefox, Opera, Mozilla (SPKAC format). On the CA side, sign the created request and send back the xxx.CRT file. As user drag and drop the xxx.CRT file to your browser and you get a dialog to trust the granted actions. Afterwards you will see the certificate as your personal certificate! Be sure that you already trusted your CA before you do this.

Export your key and keep it safe for security reasons.

There is a mozilla certutil available within the NSS package which will assist in debugging.
<http://www.mozilla.org/projects/security/pki/nss/>

Some additional infos about mozilla security modules:

<http://fisheye5.cenqua.com/browse/~raw,r=1.1/glassfish/admin-gui-ee/admin/src/java/com/sun/enterprise/ee/tools/admingui/docroot/help/ablrf.html#ablrg>

3.2.1.2 Internet Explorer

First of all you need to enable and trust your CA. With your browser, point to the address of the certificate and save the file on your disk. Double click on the file and the Certificate Installation wizard will start. Because the certificate is self signed, Internet explorer will automatically install it in the Trusted root Certificate Authority list. From now on, Internet Explorer won't complain and any Certificate signed with this root CA Certificate will be trusted too.

You can also open it from Internet explorer which will display the certificate. Click on the button Install Certificate to launch the Certificate Installation wizard.

Go to your web page and follow the certification request page for Internet Explorer (Xenroll format). On the CA side, sign the created request and send back the xxx.CRT file. As user drag

and drop the xxx.CRT file to your browser and you get a dialog to trust the granted actions. Afterwards you will see the certificate as your personal certificate! Be sure that you already trusted your CA before you do this.

Export your key and keep it safe for security reasons.

Export as well just your public key (you can do that by not exporting the private key) and publish it on your web page. This is can be used by other users who want to send you encrypted e-mails. They need your public key. Or you could first send an e-mail just with your signature (which contains as well the public key). As you like.

Microsoft provides its own certutil as part of the Windows Server administration pack. Look on google for it. You can install the admin pack on your XP desktop and then copy certadm.dll, certcli.dll, certreq.exe and certutil.exe from system32 to your home directory and afterwards you can again remove the admin pack and just use the certutil.exe directly.

3.3 Securing E-mails

You can secure your e-mails by signing and encrypting. You can use a X.509 based certificate or you can use PGP which is enabled by additional plugins. Usage of PGP is described in chapter Securing your data with PGP.

3.3.1 Generate and use an s/mime certificate

Simply generate and sign a certificate request with a valid emailAddress field and the Common Name (CN) with your real name. If you have another openssl.cnf setup, this could be different (it could be possible that you need to have the email as your CN, but not in my setup!). The important thing for the newest mail client is that you have your emailAddress field within the Subject (also called DN – distinguished name from X500/LDAP).

Now sign your message test.txt (output test.msg) using your certificate newcert.pem and your key newreq.pem:

```
openssl smime -sign -in test.txt -text -out test.msg -signer newcert.pem -inkey newreq.pem
```

You can now transmit test.msg to anybody, you can use this procedure to make signed advisories, or other signed documents to be published digitally.

3.3.2 Using Google Mail

Google mail does not provide directly a S/MIME functionality, but with some plugins you can use S/MIME with your google mail address. Of course if you use a standard mail client as well, which provides S/MIME (see Thunderbird as example).

3.3.3 Mozilla Thunderbird

You can easily use your certificate from Firefox. Just open your Account Settings and choose the Security item. You can select a certificate for signing and encrypting – it is recommended to use the same for both!

If you create a new mail message press the Security button and choose to sign or encrypt.

3.3.4 MS Outlook

You need to import your certificate as a pkcs12 file. Get the pkcs12 file by exporting your key from firefox or by conversion (14).

In MS Outlook go to Tools, Options and Security, Click on the import/export button select to

import the p12 file, enter the export password and the Digital ID "Your Name" and click on OK. Now click on the Settings button, MS Outlook should have selected the default setting so just click on New. And finally click on OK, except if you want to change the default settings. You are ready to send signed e-mails. When you send a signed e-mail the user at the other end will receive your public key, and will therefore be able to send you encrypted e-mails.


As you have issued this certificate from a self-signed certificate (root CA Certificate), the trust path won't be valid because the application does not know the root CA Certificate. The root CA certificate has to be downloaded and installed. Refer to the chapter "Install the CA root certificate as a Trusted Root Certificate in Internet Explorer".

You can send your message as encrypted signed messages or clear text message. The encryption is not really an encryption as the message contains everything needed to decrypt the message, but it ensures that the recipient won't read the message if he does not have an s/mime compliant reader.

Import certificates within outlook 2000 as file ending .p7c, no other formats supported in this version.

Assigning your Certificate to your email account:

- Open Outlook
- Select Tools from menu
- Select Options from drop down menu
- In dialog box that appears select Security tab
- Enter a name for your security setting into the Security Settings Name box
- Ensure S/MIME is selected on the Secure Message Format box
- Check the Default Security Setting for this Secure Message Format
- In Certificates and Algorithms section click the Choose button in the Signing Certificate section

Now you can sign and / or encrypt mail messages. Use the Sign  button or the Encrypt  button accordingly.

3.4 Securing your data with PGP

The used implementation is the Gnu Privacy Guard (<http://www.gnupg.org/>). With the Microsoft Windows port (<http://www.gpg4win.org/>) you can import your openssl generated certificates by using a GUI front end called Kleopatra. You can't publish your X509 keys to a keyring server, but you can use it as any other pgp key.

You can import your .p12 or PEM certificate file into kleopatra to read your openssl key.

3.5 Securing shell and file transfer

3.5.1 Openssh

Openssh is the defacto standard for secure shell usage on many linux and mac os x based systems. With it's open architecture it is ported to many other platforms even to mobile os's.

On the openssh server please setup everything so that password and host authentications are disabled and only public key authentication is enabled. You need a public key of your private key in your .ssh/ directory. As openssl will not provide you with the openssh public key format, you go through putty with the tool puttygen. In the next chapter (20) you will find information how to import your certificate with private key into putty. Start puttygen and load your key. Export it under Conversions into openssh. Copy this key (let's assume you name it openssh.pub) to your openssh server and put it to .ssh/. Run

YAPKI - How to make your own PKI system with OpenSSL

```
ssh-keygen -i -f openssh.pub > id_rsa.pub
cat id_rsa.pub >> authorized_keys
Your openssh server is now ready.
```

3.5.2 Putty

Normally you would generate your public/private keys through puttygen or getting it from another ssh tool, however it is much more easier to use your private key from your CA. To do this you need to get your RSA private key of your certificate as a pem format file.

```
openssl pkcs12 -in YOURCERT.p12 -nocerts -nodes | openssl rsa > YOURCERT-PRIVATEKEY.ssh2
```

The RSA private key PEM file should look like (this is not a valid key just an example format):

```
-----BEGIN RSA PRIVATE KEY-----
MIICWQIBAAKBgQCR9X4F4V5k6FZ70naHiVjmhevF50vHzv9t1XHS5+mcoNhTm1ln
UrTzdsYlZxgqRhWGF86RdcSESJKGFKngNa/ZZ7JFhsX2PaiQL0ldv+QaXdaycWkJ
si4wBeTdJF6WvAnS5/3HLfh6YSh4XBCp9A0RJQxaC0XrvzpJCeDEb/EhcwIBJQKB
gFbJUdn7n+j42XoOb/2P76upkxu59bUTS8TEGipgYWQMnE1jTetF7w1NipMEjKOD
ofYGzdny/znl9kHixd9ChDso7kNfLmUz1qt4cTKjOLXHxHQIFRMyMqZ63aa66zkv
9C6OA5Qy9aPxc1W88RuZZ2+3uWWhdKsh8qpIOsNB9x59AkeA77QX8mN/XzsPKLtX
owzNN8wOh3vPrHrFqd9mxVAXCOmtTbkawVQ0apa80c67DUfMaj3pGBtzzFU3guuq
0Rma5QJBAJvh07dOnh6Ed+gt09vJYT7L4MUPmbzp3gG/Owlrn4VikXzhkEV7g7Hy
q1ijspHwpaM2eo3IxA6IKRGSV+PKDXcCQHsXSpE6AyMQfWfxgAeeyj8/RbsxvauY
/bgYxhJLuM05udvNwaGMGuqgbsyal2evP3TTrwxhH8WvOC55BLCNRSkCQH5kE3II
ZIdyU2JQoq8b2Ttj4VWGCRU0QpiruFDRlGY6+n3oC/1ruhnSbPtI/id4dEK3ZJRkd
peJE4wdUD+kqXfECQDx0jjcCPCm8+AL9sDx8mPLldBT1lw1mujfq6Zl6EYywY/+p
rKwDS5YXxWgVoM3n576tKqPhTmjwPIAsYERxttiI=
-----END RSA PRIVATE KEY-----
```

Probably you have some row before -----BEGIN.... - just delete them and save the file as needed.

This key can be imported to putty by puttygen → conversions → Import Keys. Afterwards save your public and private key in putty format or even export to other ssh2 implementations.

To connect now to your openssh server with your public key on, start first the putty agent and load your private key (the putty private key format). If you don't start putty agent first, you could get errors like "Disconnected: No supported authentication methods available".

3.6 Securing a VPN

L2TP/IPSec is considered to be more secure than PPTP. If you ever have the chance to choose between L2TP/IPSec and PPTP, you should use L2TP/IPSec. However, there are times when you may want to avoid using L2TP/IPSec. The most common reason for preferring PPTP over L2TP/IPSec is when VPN clients need to connect to your VPN server while the client is behind a NAT device.

NAT devices "break" IPSec unless special measures are taken to encapsulate the IPSec packet. Encapsulating IPSec packets with a UDP or TCP header is referred to as "IPSec NAT Traversal" or NAT-T. The problem is that NAT-T is implemented in a number of different ways. Many vendors use their own proprietary NAT-T implementations. Windows Server 2003 and the Microsoft L2TP/IPSec VPN client support IETF NAT-T guidelines which are expected to become full RFC Internet standards in the near future.

Many persons consider PPTP to be an unsecure VPN protocol. This misperception is due to problems related to the initial release of PPTP (PPTP version 1). The problems with the initial release of PPTP was not so much related to the VPN protocol itself as it was the PPP authentication protocol – MS-CHAP version 1. The current version of PPTP has proven itself to

be quite secure when complex passwords are used. Both Windows 2000 and Windows Server 2003 support PPTP version 2, which is based on MS-CHAP version 2.

The level of security provided by PPTP is highly dependent on password complexity. In an ideal world all of our users will have highly complex passwords that they change everyday. We don't live in an ideal world and even when you force password policies that encourage using complex passwords, users find ways around them.

EAP-TLS allows you to get around the password complexity issue. EAP-TLS is an extension to traditional PPP authentication protocols and allows vendors to "plug in" more advanced methods of PPP authentication. EAP-TLS allows your users to log in without requiring a user name or password. VPN users obtain a user certificate and use this certificate to log into the VPN. The certificate can even be located on a "smart card" or on the user's computer.

Mobile devices have support for PPTP and as many private users have the issue with NAT, it is recommended to setup a PPTP using EAP-TLS with generated certificates. See <http://poptop.sourceforge.net/> for more information. There are patches to enable EAP-TLS on poptop. Some NAS vendors provide as well PPTP, checkout their web pages for further information (e.g. QNAP).

3.7 Securing Code

3.7.1 Microsoft Code

You can sign your programs and applet to certify that you are the author of such code. It is important for your customers to trust that nobody has tried to insert a virus or a backdoor inside your code. To authenticate your code you need Microsoft Authenticode SDK. You can get it from the Microsoft web site in the MSDN section.

You get a file called newcert.p12 that you import in the Certificate store by clicking on the file when in Windows.

You can now use this certificate for signing your code

```
signcode -cn "ACME Software cert" -tr 5 -tw 2 -n "My Application" -i  
http://www.acme.com/myapp/ -t  
http://timestamp.verisign.com/scripts/timestamp.dll myyafra.exe
```

When you try to install and run your application a dialog will appear with the title "My Application" and with a link pointed by the -i argument.

3.7.2 Java Code

Use the Java tool "keytool" to operate with certificates. Search on the internet for "java keytool openssl" and you will get all needed information.

The keytool can be used to generate the certification request to be signed by your openssl based CA. After signing and getting a PKCS#12 you can import your certificate into the java keystore with the keytool.

3.8 Securing Internet Protocols.

3.8.1 Using a certificate with mod_ssl in apache

See chapter Securing your web server.

3.8.2 Using a certificate with IMAPS / POPS / SMTP

Create a certificate as described for a web server.

The setup depends on your software solution. In case of postfix SMTP see http://www.postfix.org/TLS_README.html. In case of IMAP / POP3 see <http://www.dovecot.org/>. The principles are the same as to secure a web server.

Secure Ports:

Incoming Ports

- Secure POP3 - port 995
- Secure IMAP - port 993

Outgoing Ports

The outgoing mail is always SMTP, whether using IMAP or POP.

- Secure SMTP - port 465

3.8.3 Using a certificate with Stunnel

The stunnel program is designed to work as an SSL encryption wrapper between remote client and local (inetd-startable) or remote server. It can be used to add SSL functionality to commonly used inetd daemons like POP2, POP3, and IMAP servers without any changes in the programs' code. Stunnel uses the OpenSSL library for cryptography, so it supports whatever cryptographic algorithms are compiled into the library.

The certificates generated through this guide can be used with stunnel. You would follow the guidelines on setting up a web server certificate. But you can use as well your private one, depending on the usage of stunnel. For server like wrapping a web server like certificate is recommended.

Further information: <http://www.stunnel.org>

3.8.4 Generate and Sign a key with Microsoft Key Manager

In Microsoft Key Manager, select the service you want to create a key for, for instance IMAP (or WWW). Use the wizard to generate a new key. Ensure that the distinguished name won't be identical to previous generated keys, for instance for the Common Name (CN) use `imap.mycompany.com`. The wizard will place the request in the file `C:\NewKeyRq.txt`. Key Manager shows a Key with a strike to indicate the key is not signed. Import this file in the OpenSSL `/var/ssl` directory, rename it to `newreq.pem` and sign the request as usual.

```
CA.pl -sign
```

The file `newcert.pem` is not yet suitable for key manager as it contains some text and the `-CERTIFICATE-` section. We have to remove the text, the easy way is to do:

```
openssl x509 -in newcert.pem -out newcertx509.pem
```

Using a text editor is also suitable to delete everything outside the `-CERTIFICATE-` section.

The `newcertx509.pem` file now contains only the `-CERTIFICATE-` section.

Export the file `newcertx509.pem` to the Computer running key Manager and while selecting the key icon in the Key Manager application, right click and click on `Install the Key Certificate`, select this file, enter the passphrase. The key is now fully functional.

4 Glossary

As PKI as such defines many acronyms this glossary shall provide the most important definitions.

Term	Description
ASN.1	Abstract Syntax Notation One
BASE.64	Base64 is a group of similar encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation
CA	Certificate Authority
CER, CRT	Certificate file extension for X.509 certificates
CRL	Certificate Revocation List contains all certificates of a CA which were revoked.
DER	Distinguished Encoding Rules, message transfer syntax specified by the ITU in X.690.
IETF	Internet Engineering Task Force – supports further developments around the Internet and hosts the PKIX work group.
ITU	International Telecommunication Union
LDAP	Lightweight Directory Access Protocol
MD5	A cryptographic hash function used for example to prove integrity of files
OCSP	Online Certificate Status Protocol – used to deliver the status of certificates
P12	PKCS#12, used to exchange public and private keys.ki/PKCS for further information.
P7B, P7C	PKCS#7, used to sign / encrypt messages, formed the basis for S/MIME.
PEM	Privacy Enhanced Mail – a BASE64 encoded DER certificate, enclosed between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----"
PFX	predecessor of PKCS#12, Personal Information Exchange, used to exchange public and private keys defined by Microsoft
PGP	Pretty Good Privacy is a community data encryption approach, mainly used in e-mails or to sign objects of any type.
PKCS	Public Key Cryptography Standards. See http://en.wikipedia.org/wiki/PKCS for further information.
PKI	Public Key Infrastructure
PKIX	IETF Work group to develop internet standards support X.509 PKI's.
SSH	Secure Shell is secured shell terminal access.
SSL	Secure Sockets Layer cryptographic protocols that provides communication security over the Internet, predecessor of TLS
TLS	Transport Layer Security is an IETF standard based on SSL
X.500	ITU standard covering electronic directory services
X.509	ITU standard covering Public Key Infrastructures

5 The YAPKI system

The YAPKI system builds on top of your operating system with openssl (<http://www.openssl.org>) software. In addition it uses a web server with perl scripting enabled for the management. For the browser client enrollment on Microsoft Windows with Internet Explorer you will need the `xenroll.dll` or starting with Vista the `CertEnroll.dll`. See the MSDN for further information on those dll's and how to get them. You will use as well related software like openssh (<http://www.openssh.org>), apache web server (<http://httpd.apache.org/>), putty (<http://www.putty.org>), Ubuntu (<http://www.ubuntu.com>), Perl (<http://www.perl.org>), gnu privacy guard (<http://www.gnupg.org/>) and others.

The source is available on <http://sourceforge.net/projects/yafra/> on the yafra project page or linked on www.yafra.org.

5.1 Default directory layout

This is the default directory layout used within this document:

`xxxx/ssl/` (root directory for openssl data files / database)

```
README.txt
CA_ROOT/
backup/
gencrl.sh
gensshcert.sh
genwwwcert.sh
openssl.cnf
renp12cert.sh
renwwwcert.sh
revoke.sh
revoke.bat
```

within `CA_ROOT` you will find the standard openssl directory layout which includes the key file "index.txt" and the root CA certificate.

5.2 YAPKI scripts

The following scripts are used to ease the administration of openssl and to enable a simple web management interface.

Script name	Description	Run environment
<code>gencrl.sh</code>	Generate certification revocation list	Unix bash
<code>gensshcert.sh</code>	Generate web server certificate	Unix bash
<code>genwwwcert.sh</code>	Generate web server certificate	Unix bash
<code>renp12cert.sh</code>	Renew P12 certificate	Unix bash
<code>renwwwcert.sh</code>	Renew web server certificate	Unix bash
<code>revoke.sh</code>	Revoke certificate	Unix bash
<code>revoke.bat</code>	Revoke certificate	Windows batch
<code>pki-mgmt.pl</code>	Simple web based management interface	Perl script

6 Document release notes

Tested installations:

- Ubuntu Server LTS 10.04 with openssl, openssh, apache2 with mod_perl and mod_ssl
- Windows XP SP2 with cygwin, XAMPP and OpenSSL 0.98a

The back end of YAPKI is based on a LTS Ubuntu Server release. Microsoft Windows is of course as well an option, but only tested once to have a proof of concept. Some scripts are available as dos batch files, so it's easy to modify the missing ones if you want to run it on Windows.

On Windows you need additional software. You could probably run it with native ports but I preferred [CYGWIN](#). Install it with cygservice, openssl and openssh. This is the UNIX layer on top of Windows (cygwin is quite compatible with most linux distributions). For apache web server I used apachefriends package.

For further information please visit <http://www.yafra.org>.